Express Framework

MINIMALIST WEB FRAMEWORK FOR NODE.JS

PROGRAMAÇÃO WEB 1

Objetivos de Aprendizagem

- Apresentar os conceitos fundamentais do *framework* Express

Agenda

- Funcionamento básico
- *Middlewares*
- Rotas

Funcionamento Básico

1. No arquivo `app.js`
    1. Importação do módulo Express
    2. Criação da variável de aplicação (app)
    3. Inicia um servidor na porta 3000 para atender requisições GET
    4. Exibe uma mensagem no console

```
const express = require('express')
const app = express()
const port = 3000


app.get('/', (req, res) => {
  res.send('Hello World!')
})
app.listen(port, () => {
console.log(`Example app listening on
port ${port}`)
})
```

Tratamento de requisição

- O servidor recebe requisições (`req`) HTTP e responde (`res`) a essas requisições (GET)

```
app.get('/', (req, res) => {
  res.send('Hello World!')
})
```

> *Routing* refers to determining how an application responds to a client request to a particular endpoint, which is a URI (or path) and a specific HTTP request method (GET, POST, and so on).

Roteamento Básico

https://expressjs.com/en/starter/basic-routing.html

Roteamento básico

- *Routing*
- **app** = instância do Express
- **METHOD** = método HTTP a ser tratado
- **PATH** = caminho no servidor
- **HANDLER** = função que será executada quando a rota for acionada

```
app.METHOD(PATH, HANDLER)
```

Exemplos

```
app.put('/', (req, res) => {
    res.send('Got a PUT
  request’)
  })
```

```
app.post('/', (req, res) => {
 res.send('Got a POST
request’)
})
```

Como realizar testes com métodos diferentes do GET?

“

Postman is an API platform for building and using APIs. It simplifies each step of the API lifecycle and streamlines collaboration so you can create better APIs—faster.

”

https://www.postman.com/

What is Postman?

POSTMAN

- *Desktop*
- *Online*

app.all()

- Utilizada para atender a todos os métodos HTTP
- O exemplo ao lado é executado para requisições de GET, POST, PUT, DELETE, etc

```
app.all('/', (req, res, next) => {
    console.log('Accessing section ...')
   next() // pass control to the next handler
})
```

Caminhos de Rotas

- São definidos através de padrões de strings ou expressões regulares

This route path will match requests to the root route, /.

```
app.get('/', (req, res) => {
  res.send('root')
})
```

This route path will match requests to /about.

```
app.get('/about', (req, res) => {
  res.send('about')
})
```

This route path will match requests to /random.text.

```
app.get('/random.text', (req, res) => {
  res.send('random.text')
})
```

Parâmetros de Rotas

- São utilizadas para passar parâmetros especificados pelas URLs
- O objeto `req.params` recebe os valores

```
Route path: /users/:userId/books/:bookId
Request URL: http://localhost:3000/users/34/books/8989
req.params: { "userId": "34", "bookId": "8989" }
```

Parâmetros de Rotas

```
Route path: /flights/:from-:to
Request URL: http://localhost:3000/flights/LAX-SFO
req.params: { "from": "LAX", "to": "SFO" }
```

```
Route path: /plantae/:genus.:species
Request URL: http://localhost:3000/plantae/Prunus.persica
req.params: { "genus": "Prunus", "species": "persica" }
```

Propriedades úteis

`request`

- Propriedades úteis
  - `req.url`
  - `req.cookies`
  - `req.ip`
  - `req.params`
  - `req.path`

`response`

- Métodos úteis
  - `res.attachment()`
  - `res.cookie()`
  - `res.end()`
  - `res.status()`
  - `res.json()`

“Middleware functions have access to the request object (req), the response object (res), and the next function in the application’s request-response cycle.”

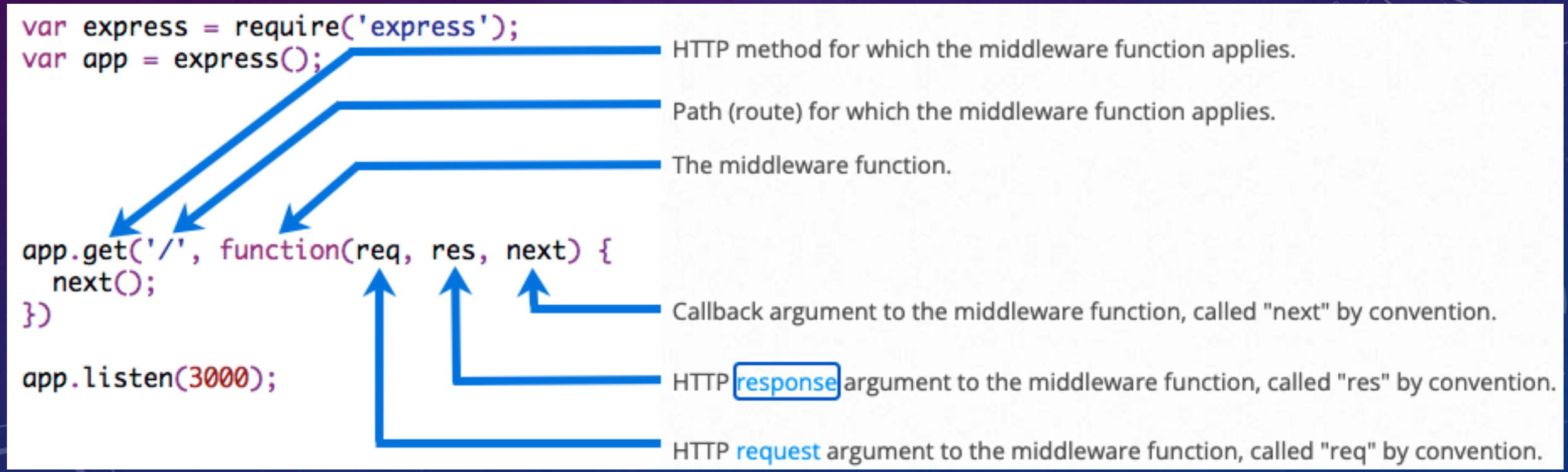
Middleware

https://expressjs.com/en/guide/writing-middleware.html

Middleware

- Funções de *middleware* realizam as seguintes tarefas:
  - Executam código
  - Modificam os objetos de requisições e/ou respostas
  - Finalizam o ciclo de requisição/resposta
  - Chamam a próxima função de *middleware* da pilha

Middleware


var express = require('express');
var app = express();
app.get('/', function(req, res, next) {
next();
})
app.listen(3000);
HTTP method for which the middleware function applies.
Path (route) for which the middleware function applies.
The middleware function.
Callback argument to the middleware function, called "next" by convention.
HTTP response argument to the middleware function, called "res" by convention.
HTTP request argument to the middleware function, called "req" by convention.

Exemplo

```
const express = require('express')
const app = express()

const myLogger = function (req, res, next) {
    console.log('LOGGED')
    next()
}
app.use(myLogger)
app.get('/', (req, res) => {
    res.send('Hello World!')
})
app.listen(3000)
```

```
const express = require('express')
const app = express()
const requestTime = function (req, res, next) {
  req.requestTime = new Date().toDateString()
  next()
}
app.use(requestTime)
app.get('/', (req, res) => {
  let responseText = 'Hello World!<br>'
  responseText += `<small>Requested at: ${req.requestTime}</small>`
  res.send(responseText)
})app.listen(3000)
```

Tipos de *Middleware*

- MW de Aplicação (*Application Level MW*)
- MW de Roteamento (*Router Level MW*)
- MW de tratamento de erro
- *Embedded*

Aplication-level Middleware

- Associadas a uma instância de `app` utilizando:
  - **`app.use()`**
  - **`app.METHOD()`**

Exemplos

```
const express = require('express')
const app = express()

app.use((req, res, next) => {
  console.log('Time:', Date.now())
  next()
})
```

```
app.use('/user/:id', (req, res, next) => {
  console.log('Request Type:', req.method)
  next()
})
```

Exemplos

```
app.get('/user/:id', (req, res, next) => {
  // if the user ID is 0, skip to the next route
  if (req.params.id === '0') next('route')
  // otherwise pass the control to the next middleware function in this stack
  else next()
}, (req, res, next) => {
  // send a regular response
  res.send('regular')
})

// handler for the /user/:id path, which sends a special response
app.get('/user/:id', (req, res, next) => {
  res.send('special')
})
```

Router Level Middleware

- Funcionamento similar aos MW de aplicação
- Rodam em uma instância de **`express.router()`** ao invés de **`express.app()`**
- Podem ser usados através de **`router.use`**() ou **`router.METHOD()`**

```
const express = require('express')
const app = express()
const router = express.Router()

// a middleware function with no mount path. This code is executed for every request to the router
router.use((req, res, next) => {
  console.log('Time:', Date.now())
  next()
})

// a middleware sub-stack shows request info for any type of HTTP request to the /user/:id path
router.use('/user/:id', (req, res, next) => {
  console.log('Request URL:', req.originalUrl)
  next()
}, (req, res, next) => {
  console.log('Request Type:', req.method)
  next()
})

// a middleware sub-stack that handles GET requests to the /user/:id path
router.get('/user/:id', (req, res, next) => {
  // if the user ID is 0, skip to the next router
  if (req.params.id === '0') next('route')
  // otherwise pass control to the next middleware function in this stack
  else next()
}, (req, res, next) => {
  // render a regular page
  res.render('regular')
})

// handler for the /user/:id path, which renders a special page
router.get('/user/:id', (req, res, next) => {
  console.log(req.params.id)
  res.render('special')
})

// mount the router on the app
app.use('/', router)
```

Tratamento de erros

- Similar aos outros tipos de MW, porém com quatro argumentos

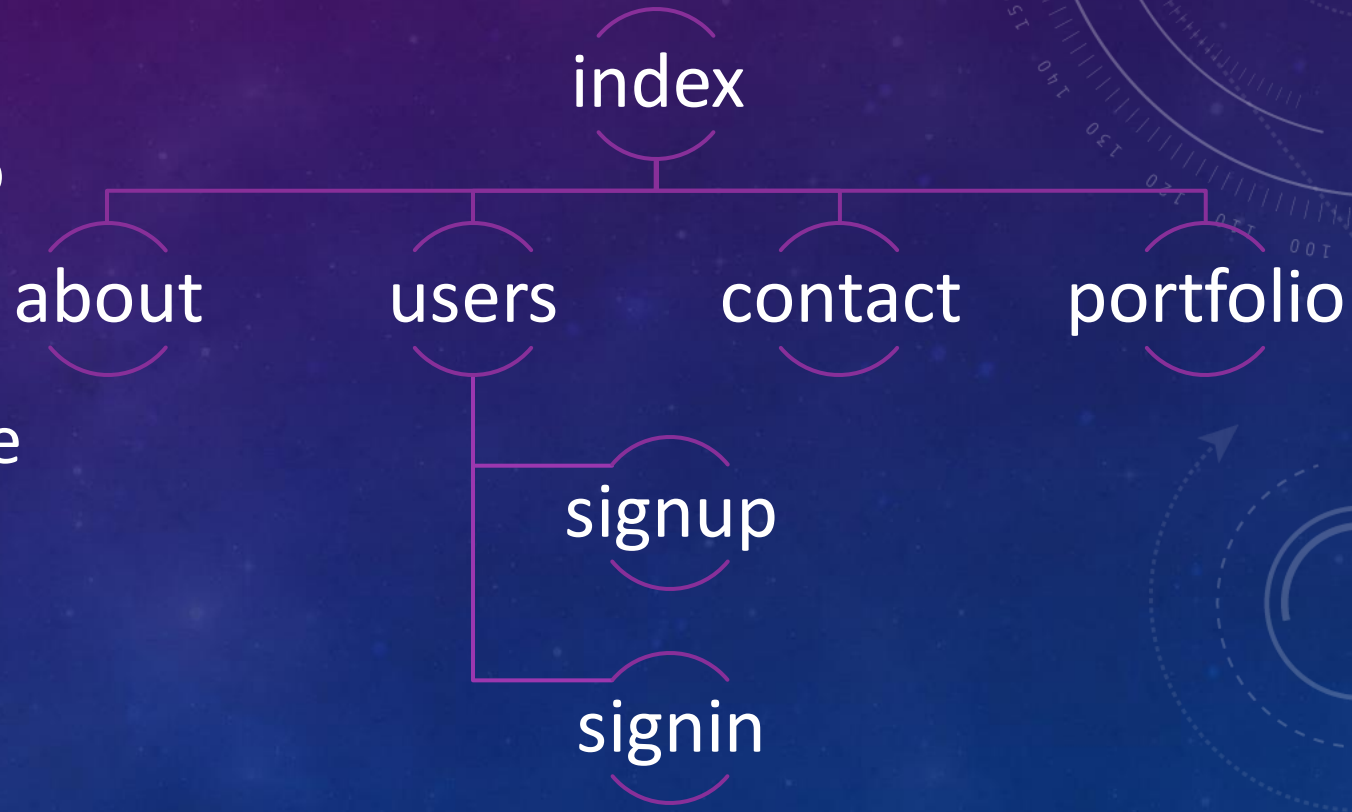
```
app.use((err, req, res, next) => {
  console.error(err.stack)
  res.status(500).send('Something broke!')
})
```

Embutidos

- **`express.static`**
  - Serve arquivos estáticos (HTML, imagens, CSS, etc)
- **`express.json`**
  - Server arquivos JSON
- **`express.urlencoded`**
  - Trata requisições com URLs codificadas

Exercício

Criar uma aplicação utilizando o *framework* Express que contenha a estrutura mínima mostrada. Cada página atende com requisições GET.



Exercício - Requisitos

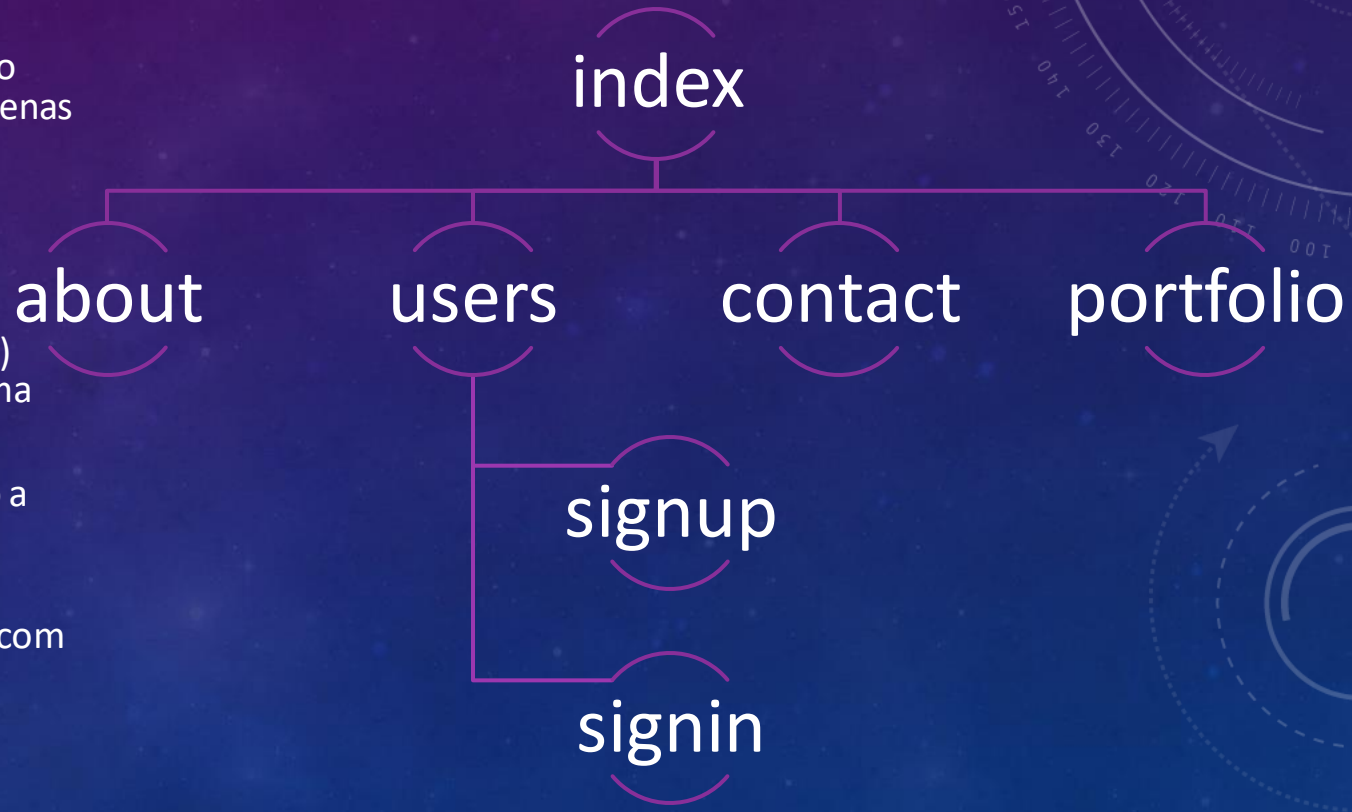
i. Criar uma função de Middleware de roteamento para cada uma das rotas mostradas exibindo apenas uma página de título

ii. Criar uma função *Middleware* de aplicação que realiza o registro no console do acesso a cada página (LOG);

iii. Em signin, criar uma rota (`/users/:userid`) que recebe o id do usuário e exibe na página uma msg de boas vindas usando esse valor;

iv. Caso o usuário acesse sem userid é direcionado a página signup;

v. Qualquer outra página que não tenha a rota indicada é direcionada para página de erro 404 com link para o index;



Referências

- https://expressjs.com/en/starter/hello-world.html
- https://www.tutorialspoint.com/expressjs/index.htm